Planning an Experiment on User Performance for Exploration of Diagrams Displayed in 2½ Dimensions

Johannes Bohnet, Jürgen Döllner

Hasso-Plattner-Institut – University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Postdam, Germany
{bohnet, doellner} @hpi.uni-potsdam.de

Abstract: Two dimensional node-link diagrams such as those proposed by the Unified Modeling Language (UML) have become an important means of communication and documentation of various aspects of software systems. When complex systems are concerned, these diagrams typically consist of a great amount of connected nodes – particularly if they origin from reverse engineering processes. In software visualization research community various papers have been published arguing that displaying diagrams in 3D or 2½D, i.e., on a plane seen under 3D perspective, helps to cope with the size of the diagrams and supports users in exploring them. However, formal experiments supporting the hypothesis that user performance increases when analyzing diagrams in 3D or 2½D instead of 2D is usually missing. In this paper we describe a formal experiment we are about to conduct to examine user performance for tool-supported exploration of control flow diagrams displayed in 2½D.

1 Introduction

Business processes of most enterprises are back-boned by long-living IT software systems. These legacy software systems have typically become significantly complex and are therefore hard to maintain. Complexity results from the size of the implementation, typically consisting of more than a million lines-of-code (MLOC). A single developer is only capable of understanding a small fraction of the system in full detail. Hence, more and more design anomalies are introduced into the system's code over time and quality declines as a software system is usually developed and maintained by a great number of actors in different roles – each actor performing changes based on an incomplete understanding of the system [PA94].

Reverse engineering techniques exist to support developers in gaining an understanding of various aspects of the software system such as structural or behavioral aspects. These techniques aim on reconstructing software models from source code representation and visualizing them. Often models known from forward engineering are used, such as those defined by the Unified Modeling Language (UML). However, even if a complete set of models existed during design time and implementation would have been created by transforming the models into code, reconstructing the models from code is only rarely possible as the syntax of programming languages such as C or C++ do not reflect important concepts from modeling stage. High-level structuring of the system for instance, is typically only implicitly contained in code. Hence, automatically

reconstructed models are incomplete, lack of important ideas from design time and are therefore much more unstructured then models created manually during design time. Visualizing these models and preparing them for being explored is a challenging task. Applying data mining techniques such as outlier detection, clustering, or data aggregation [KH05] using dependency information between software artifacts or other software metrics [LK94] provide means of further structuring and simplifying the underlying model. However, these techniques only propose simplifications that need to be checked manually, so visualization still needs to cope with a large number of model elements.

In state-of-the-art software engineering tools software models are typically visualized as 2 dimensional node-link diagrams that can be explored interactively. Various research groups have proposed to make use of the additional third dimension for diagram visualization to cope both with the limited display space and with the problem of crossing arcs [e.g., FD98, IC03, LS02, PLL05, TMR02]. Ideas are driven by extended possibilities of layouting nodes or choosing shape geometries for node representation. However, formal experiments that prove enhanced user performance when visualizing in 3D instead of 2D is typically missing.

In this paper we describe a formal experiment that we are about to conduct, which aims on measuring user performance of software developers exploring node-link diagrams that are displayed in 2½D, i.e., on a plane seen under 3D perspective, and represent control flow graphs captured from complex software systems during runtime. The experimental setup reflects a small fraction of a tool's process for locating features in unfamiliar code of complex software systems, which has been developed by our research group. Details on the tool for feature location in +MLOC systems can be found in [BD06].
The objective we are investigating is: "Is user performance for exploration of control flow diagrams increased when displaying diagrams in 2½D instead of 2D?"

2 Experimental Environment

Formal experiments require high level of control over variables that can affect the truth of the stated hypothesis. Therefore, we restrict the experiment to only a very specific, basic exploration task that is needed to be performed during feature analysis with our tool. Furthermore, we restrict our experiment to only one specific type of diagram representing control flow graphs. For reasons of clarity, Figure 1 illustrates the feature location tool's (simplified) analysis process. First, a user semi-automatically extracts the module structure of the complex software system by analyzing the source code's directory structure. Second, the user runs the software system in a way that the specific system's feature, which is to be located within the code, is executed. During execution the analysis tool captures function calls within the software system and in collaborating systems. The resulting call graph is then related to the system's module structure, i.e., functions are assigned to modules.

Finally, the user picks a starting function and explores the part of the call graph that consists of functions closely related to the chosen function via calls. For this, the

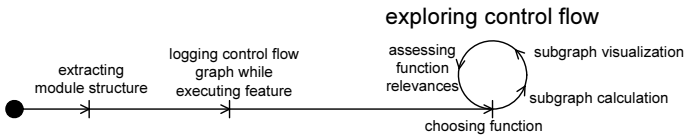

Figure 1: Analysis process with our tool for feature location.

subgraph is visualized as node-link diagram; boxes represent functions, arrows represent function calls, and large boxes represent modules. Module representations can be nested, i.e., they may contain other module or function representations. The layout is calculated by *dot*, which is a part of the *Graphviz* tool suite (*www.graphviz.org*). During exploration a user may consider a specific function as interesting. Then she/he chooses this function, a new subgraph is extracted from the complete graph, and the subgraph is visualized for exploration thereupon. This loop enables users to explore the complete function call graph and to locate the feature's core implementation in a step-wise way.

However, the user's major activity is to be performed during exploring a visualized subgraph. He/she needs to asses each function's relevance for the analyzed feature. After this assessment, the user is able to choose the most relevant function and triggers the calculation of a new subgraph. The user's assessment whether a function is relevant for a feature or not is based on 4 criteria:

1) the function name;
2) the name of the module of which the function is a part;
3) call relations between functions, i.e., whether a function is directly or indirectly called by a function that has previously been classified as relevant;
4) call relations on module level, i.e., whether a function is part of a module previously classified as relevant and the function is called by another function being part of a different module.

2.1 The Visualization System

The visualization system, which provides means of interactive diagram exploration, is based on the *Virtual Rendering System* (*www.vrs3d.org*), which is an OpenGL supporting 3D graphics library. VRS provides enhanced 3D navigation techniques and an elaborated labeling mechanism for attaching textual information to 3D objects. As the visualization system of the feature location tool is based on VRS and both display modes, 2D and 2½D, are seamless obtained from 3D computer graphics (see below), navigation and labeling aspects represent state variables of the experiment, which are hold constant. This is a prerequisite for being able to compare results of 2D and 2½D mode.

Implementing 2D diagram visualization using a 3D graphics system

As illustrated in Figure 2 (left) the visualization system of our tool for feature location uses a 3D virtual environment to implement 2D diagram visualization. Boxes representing functions and arrows representing calls are painted on a 2D layer, which is placed in 3D world. Several other layers for module representations are closely

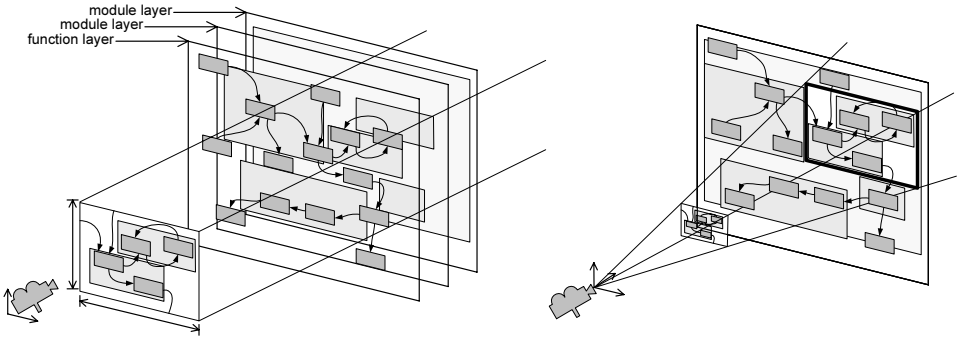

Figure 2: left: 2D diagrams are created by using orthographic projection on various 3D layers. right: For 2½D visualization, projection is changed from orthographic to perspective. The resulting image is only slightly changed if the virtual camera looks straight down onto the layers.

positioned behind this layer. Finally, this 3D scene is reduced to 2D by orthographic projection, i.e., a virtual camera looking straight onto the layers is positioned in front of the layers and defines the cutout of the layers that is displayed on the screen. Changing camera parameters such as position and the height and width of the cutout area enables users to navigate and to explore the whole diagram. Camera position controls what part of the diagram is in focus. Height and width of the cutout area define the zooming level.

Continuous transition from 2D to 2½D diagram visualization

The transition from 2D to 2½D diagram visualization requires an initial change from orthographic projection to perspective projection, i.e., a change in specification of the virtual camera, which defines the cutout area of the layers (Figure 2). For 2D visualization, the 3D subvolume that is being rendered to an image is a cuboid defined by camera position and height and width of the cuboid front. For 2½D visualization, the 3D subvolume is a frustum defined by a different camera specification, namely the camera position and the angles that open the pyramid in height dimension and in width dimension. These angles, also specified by parameters known as *field-of-view-angle* and *aspect ratio*, are set to constant values. Hence, the cutout of the layers is only determined by the camera position. The position in height and width dimensions controls, which part of the diagram is in focus, the position in depth dimension, i.e., the distance from the layers, defines the zooming level.

The change from orthographic to perspective projection causes slightly different images. Elements shown near the image border are distorted. That's why the first step of our experiment needs to be a test whether the change in projection type affects user performance. Our hypothesis is that specific camera specification values exist so that user performance is not affected.

With perspective projection, we are able to continuously transition to 2½D visualization (Figure 3). Tilting the viewing direction of the virtual camera, away from its orthogonal direction in respect to the layers, results in a perspective view. Elements in the front, i.e., lower part of the image, are now shown in full detail whereas elements in the back, i.e., in the upper part of the image, are shrunk and provide overview-like information only.

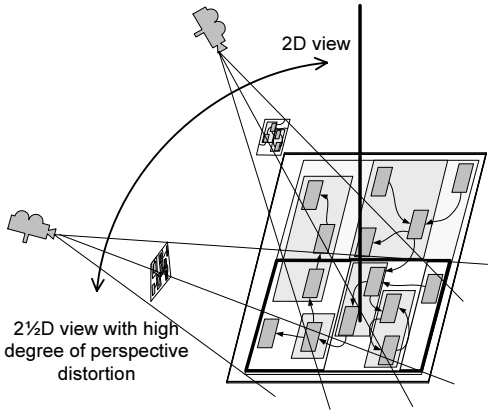

Figure 3: Tilting the viewing direction of the virtual camera, continuously changes diagram visualization from 2D to 2½D with increasing perspective distortion.

The more the viewing direction is tilted, the more elements are shown at the same time, albeit most of them being distorted and displayed with low detail.

Navigation techniques

The navigation techniques for exploring the diagram, i.e., determining the currently visible part, are similar to navigation techniques known from standard office applications:

Pane Navigation:
: Dragging the mouse while holding the left mouse button pressed, enables users to move the diagram accordingly to the mouse movement.

Zoom Navigation:
: Using the mouse wheel while the control key is pressed determines the zooming level.

Move-Focus Navigation:
: Clicking on a specific position on the diagram while the control key is pressed animates the diagram such that the chosen position is in the center of the screen.

Annotating text to diagram elements

For diagram exploration it is important to be able to relate visualized geometries, such as boxes or arrows, to the underlying data, such as functions, modules, and calls. For this, our visualization system provides a technique for annotating 3D objects with textual information such that text occlusion is minimized. A detailed description of the labeling mechanism can be found in [MD06]. In the context of our experiment, it is important to note that the mechanism works for 2D and 2½D in the same way: Depending on the size that an element, e.g., a box representing a function, occupies on the screen, the element is equipped with a text label, e.g., showing the function name. If the size on the screen decreases a critical threshold, the labeling mechanism is turned off for this element. This way, user interaction is equal for both 2D and 2½D mode: To determine the function or

module name of a displayed box shape, a user needs to zoom in until the box is shown in a size greater than threshold.

3 Experiment Design

The objective we are examining is: "Is user performance for exploration of control flow diagrams increased when displaying diagrams in 2½D instead of 2D?"
Therefore we state the null hypothesis: "There is no difference in user performance for exploration of node-link diagrams representing control flow graphs when 2D visualization and when 2½D visualization is used."
The alternative hypothesis reads as: "User performance for exploration of node-link diagrams representing control flow graphs is increased when using 2½D visualization compared to 2D visualization."

For reasons of clarity, we shortly recapitulate the main activity that needs to be performed with our tool for feature location, and then describe how experimental tasks are deduced. As explained in Section 2, the user's main activity consists of assessing a function's relevance for locating and understanding a specific feature of the software system under analysis. This assessment is based on 1) the function name, 2) the module name of which the function is a part, 3) call relations between functions, and 4) call relations on module level.

In our experiment the subjects shall need to perform the exploration task necessary for the above mentioned assessment task. However, the assessment itself is replaced by a trivial task based on naming conventions and is free of existing knowledge on the software system under analysis. Hence, the experiment is independent of the subjects' reverse-engineering skills and experiences. The experimental task is to explore a given subgraph and assign 0 to 4 points to each function. The points correspond to a trivial assessment task defined by the following rules:

1) initially every function has 0 points;
2) a function whose name is ending with a vowel gets 1 point;
3) a function fulfilling rule 2 and being part of a module whose name is ending with a vowel gets 1 point;
4) a function fulfilling rule 2 and being directly or indirectly called by another function, which fulfills rule 2, gets 1 point;
5) a function gets 1 point if it is part of a module whose name is ending with a vowel and if it is being called by a function being part of a different module.

The experimental subjects participating in the experiment are approximately 100 university students of software engineering faculty, who will be asked to give a self-evaluation on their experiences with 2½D visualizations beforehand. Before performing the experiment they are trained on using the visualization system. The diagrams that the experimental subjects are working on are extracted from real-world situation, i.e., they are control flow graphs of the *Mozilla Firefox Browser (www.mozilla.org)* extracted from executing typical Firefox features. However, function and module names are modified so that the primitive assessment task can be performed based on vowels at the end of the names.

State variables of the experiment that are varied are:
- number of functions in a diagram (30, 50, 70, 90)
- tilting angle of viewing direction of virtual camera, i.e., degree of perspective distortion (90° = 2D, 75°, 60°, 45°)
- major flow direction of control flow diagrams: →, ↑, ← or ↓

The subjects are given a fixed time period to complete the task of exploring a diagram and of assigning points to each function shown. The response variable we are investigating is the number of errors per task.

Before the experiment is started, a pre-experiment is conducted to clarify the hypothesis: "A reasonable specification of virtual camera exists so that there is no statistically significant difference in user performance for exploration of node-link diagrams representing control flow graphs when 2D diagram visualization is implemented by orthographic projection and by perspective projection, respectively." This pre-experiment is necessary to be able to compare results obtained from 2D visualization implemented by orthographic projection with results obtained from 2½D visualization implemented by perspective projection.

4 Summary

In this paper we describe an experiment we are about to conduct to examine the question whether the use of 2½D visualization techniques increases user performance for exploration of simple node-link diagrams representing control flow graphs, compared to the use of common 2D diagram visualization.

References

[BD06] J. Bohnet, J. Döllner. Visual Exploration of Function Call Graphs for Feature Location in Complex Software Systems. *Proc. of ACM Symp. on Software Visualization,* 2006, pp. 95-104.

[FD98] L. Feijs, R. DeJong. 3D visualization of software architectures. *Communications of the ACM*, 41, 12, 1998, pp. 73-78.

[IC03] W. Irwin, N. Churcher. Object Oriented Metrics: Precision Tools and Configurable Visualisations. *Proc.of Int'l Symp. on Software Metrics,* 2003, pp. 112-123.

[KH05] S. M. Kocherlakota, C. G. Healey. Summarization Techniques for Visualization of Large, Multidimensional Datasets. *Tech. Report TR-2005-35,* 2005.

[LS02] C. Lewerentz, F. Simon, Metrics-Based 3D Visualization of Large Object-Oriented Programs. *Proc. of Int'l Workshop on Visualizing Software for Understanding and Analysis,* 2002, pp. 70-77.

[LK94] M. Lorenz, J. Kidd. Object-oriented software metrics. *Prentice-Hall,* 1994.

[MD06] S. Maass, J. Döllner. Efficient View Management for Dynamic Annotation Placement in Virtual Landscapes. *Proc. of Int'l Symp. on Smart Graphics,* 2006, pp. 1-12.

[PA94] D. L. Parnas. Software Aging. *Proc. of Int'l Conf. on Software Engineering,* 1994, pp. 279-287.

[PLL05] T. Panas, R. Lincke, W. Löwe. Online-configuration of software visualizations with Vizz3D. *Proc. of ACM Symp. on Software Visualization,* 2005, pp. 173-182.

[TMR02] A. Telea, A. Maccari, C. Riva, An Open Visualization Toolkit for Reverse Architecting. *Proc. of Int'l Workshop on Program Comprehension,* 2002, pp. 3-10.